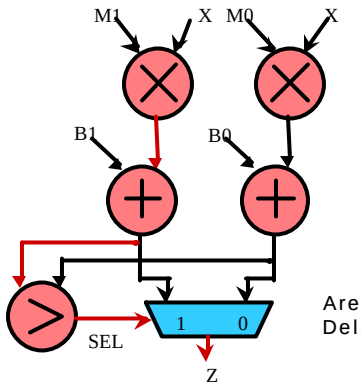
MAX0 - simple, binary



Area ~= 5 Propagate Adders
Delay ~= 3 Propagate Adders

```
wire   [2*n]   P1 = M1 * X;
wire   [2*n]   P0 = M0 * X;
wire   [2*n+1] Y1 = P1 + B1;
wire   [2*n+1] Y0 = P0 + B0;
output [2*n+1]  Z = (Y1>Y0) ? Y1 : Y0;
```

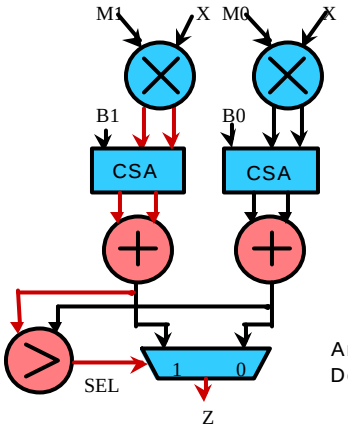
With Module Compiler there is always a **one-to-one correspondence** between the input MCL code and the architecture of the synthesized circuit.

Each operator maps directly to a built-in gate level micro architecture.

This is **architectural synthesis**.

This design compares two different multiply and add circuit outputs and selects the maximum of the two. It has 5 carry propagate structures. Each multiplier has a carry propagate adder to form its binary output, there are two adders, and one comparator. That makes a total of 5. There are three carry propagations in series on the critical path.

MAX1 - multiply and add in carrysave



Note: csa is simplified for concept only

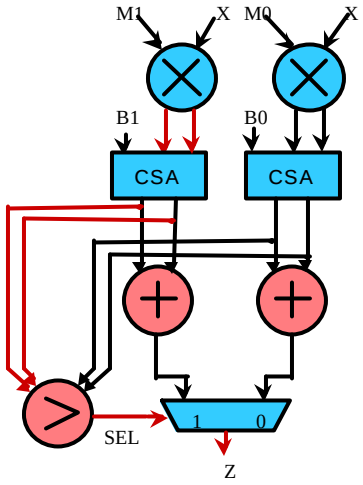
Area ~= 3 Propagate Adders
Delay ~= 2 Propagate Adders

```
wire   [2*n+1] Y1 = M1*X + B1;
wire   [2*n+1] Y0 = M0*X + B0;
output [2*n+1]  Z = (Y1>Y0) ? Y1 : Y0;
```

By writing the multiply-add expressions on one line we are exploiting MC's operator merging feature. Operator merging with MC eliminates the propagate adder of the multipliers.

This leaves us with 3 propagate adders in the design with the critical path containing 2 of them. We should expect a reduction in both delay and area than the previous case (MAX0)

MAX2 - multiply, add, and compare in carrysave



Note: csa is simplified for concept only

Area ~= 3 Propagate Adders
Delay ~= 1 Propagate Adders

```
directive (Carrysave=cs);
wire   [2*n+1] Y1 = M1*X + B1;
wire   [2*n+1] Y0 = M0*X + B0;
directive (Carrysave="off");
wire   [2*n+1] Z1 = Y1; // binary
wire   [2*n+1] Z0 = Y0;
output [2*n+1]  Z = (Y1>Y0) ? Z1 : Z0;
```

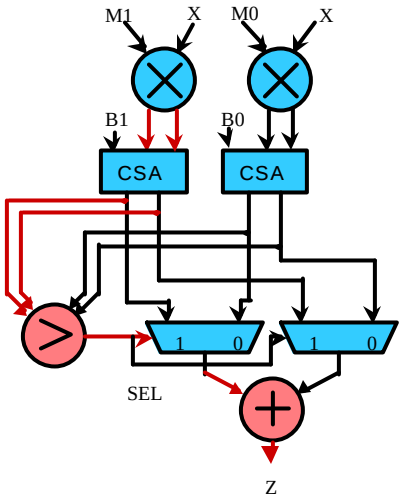
This version is possible only in MC. The only thing we have done differently here is passing the multiply-and-add outputs in carry save form directly to the comparator before reducing them to binary.

We still have 3 propagate adders in the design. But we have moved them such that the delays of each occur in parallel by putting just 1 propagate adder in the critical delay. The multiply-add results are resolved to binary before the multiplexor. This occurs at Z1 and Z0.

Notice that the comparator in MC can accept inputs which are not even binary! In the expression **`Y1>Y0`**, both Y1 and Y0 are in carrysave format.

As a rule, always try to figure if there is any way to further reduce the number of propagate adders in a design. This will usually, buy either timing, area, or both.

MAX3 - mult, add, compare, and mux in carrysave



Note: csa is simplified for concept only

1. This architecture is for area improvement only.

2. To code this example, you must have an individual access to the carry and sum terms to mux them.

Area ~= 2 Propagate Adders
Delay ~= 2 Propagate Adders

```
directive local (carrysave="convert");
wire   [2*n+1] Y1 = M1*X + B1;
directive local (carrysave="convert");
wire   [2*n+1] Y0 = M0*X + B0;
wire [2*n+1] Y1s, Y1c, Y0s, Y0c;
csconvert(Y1s,Y1c,Y1);
csconvert(Y0s,Y0c,Y0);
wire [2*n+1] Zs = (Y1>Y0) ? Y1s : Y0s;
wire [2*n+1] Zc = (Y1>Y0) ? Y1c : Y0c;
output [2*n+1] Z = Zs + Zc;
```

The present architecture is strictly for area improvement. The speed is not the main criterion here. (This design is actually slower than the previous case!!)

In this design (MAX3) all we have done compared to the previous case is now we actually pass the carry save terms through the muxes and then use just a single propagate adder to reduce the carry and sum terms to binary. Now, we can easily see that there is only a total of only 2 propagate adders in the design, so we expect an area improvement.

But, when we look at the critical path, we see that both of these propagate adders are in the critical path. So this will be slower than the previous example.

In this final case, using bit level access of the carry save terms did not buy us performance but it did give us the smallest design of all the architectures for this technology library, and if the clock is slower than 100 MHz, then this would be the optimal design.